SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

LEVEL II

12 B.S.

1. REPORT NUMBER: HPP80-26 ✓	2. GOVT ACCESSION NO.: AD-A096511
4. TITLE (and Subtitle): (6) The Nature of Heuristics	5. TYPE OF REPORT & PERIOD COVERED: (9) Technical rept.
	6. PERFORMING ORG. REPORT NUMBER: ---
7. AUTHOR(s): (10) Douglas B. Lenat	8. CONTRACT OR GRANT NUMBER(s): (15) N00014-80-C-0609 ✓
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Department, Stanford University, Stanford, California 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (11) 29 Dec 80 ---
11. CONTROLLING OFFICE NAME AND ADDRESS: Mathematical & Informations Sciences Div. Office of Naval Research, 800 No. Quincy Street, Arlington, Va. 22217	12. REPORT DATE: December 29, 1980
	13. NUMBER OF PAGES: 30 (12) 31
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): (14) HPP-80-26	15. SECURITY CLASS. (of this report): ---
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: ---

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Accession For
NTIS GRA&I X
DTIC TAB
Unannounced
Justification
By Per Ltr. on File
Distribution
Availability Codes
Dist | Avail and/or Special
A

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

---

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Discovery, Learning, Induction, Meta-rules, Heuristics, Knowledge, Expert Systems, Representation.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DTIC ELECTE MAR 18 1981 S D

D

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

JOB

094120

81 2 18 004

AD A096511

DTIC FILE COPY

The Nature of Heuristics

Douglas B. Lenat[1]

December 29, 1980

The bottleneck of building expert systems is knowledge acquisition, and one long-range solution is for the program to learn via discovery. New domains of knowledge can be developed by using heuristics, yet as they emerge *new* heuristics are needed. They in turn can be discovered by using a body of heuristics for guidance. How exactly does this process work? Must there be a separate body of "meta-heuristics"? How intertwined are heuristics with Representation of knowledge? In trying to find new heuristics, is it cost-effective to try to improve the existing representation of knowledge, and if so how can this be automated? What is the *nature* of heuristics, their "first-order theory"? What are the implications of such a theory upon the design of a program which discovers new heuristics? These questions are among those that our research -- and this paper -- address.

1. MOTIVATION

Several recent programs in Artificial Intelligence (AI) perform complex tasks demanding a large corpus of expert knowledge [Feigenbaum 77]. Consider, for example, the PROSPECTOR program for evaluating the mineral potential of a site, the MYCIN program for medical diagnosis, and the MOLGEN program for planning experiments in molecular genetics. To construct such a system, a knowledge engineer talks to a human expert, extracts domain-specific knowledge, and adds it to a growing knowledge base usable by a computer program (see Fig. 1). The critical stage of this process, the limiting step, is the transfer of expertise. From the program's point of view, the limitation is the slow rate at which it acquires knowledge. This is the central problem facing knowledge engineering today, the bottleneck of knowledge acquisition.

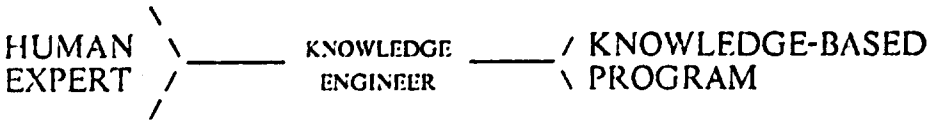

Figure 1: The bottleneck of knowledge acquisition is *transfer of expertise*. This comprises (*i*) the expert's difficulty in articulating what he knows, and (*ii*) the impedance mismatch between the concepts and vocabulary of the expert and the knowledge engineer.

Two possible solutions to this problem suggest themselves (though they are not mutually exclusive.) First, one might try somehow to widen the channel joining expert to program, for example by building a sophisticated natural language interface.

The difficulty with this is that the expert must communicate not merely the "facts" of his field, but also the heuristics: the informal judgmental rules which guide him. These are rarely thought about concretely, and almost never appear in journal articles, textbooks, or university courses. Thus, even with a wider channel, the expert would have difficulty in verbalizing his heuristics.

[1] The author is an asst. professor of computer science at Stanford University, Stanford, Ca. 94305.

The second possible solution is to sever the umbilicus entirely: eliminate the knowledge engineer and the human expert, expose the program to the environment, and let it discover new knowledge on its own. Can this be done? Since *knowledge* comprises both facts and heuristics, the question divides into two parts: can new domain concepts and relationships be discovered, and can new domain heuristics be discovered? This paper is addressed to these questions, and it presents evidence that the answers are affirmative.

Along the way, an elementary "theory of heuristics" accrues. Our initial definition of a heuristic is: a piece of knowledge capable of suggesting plausible actions to follow or implausible ones to avoid. In Section 3, it becomes apparent that this is insufficient: for a body of heuristics to be effective (useful for guiding rather than merely for rationalizing in hindsight) each heuristic must specify a situation or context in which its actions are especially appropriate or inappropriate. The theory developed in Section 5 is based on this definition.

2. OVERVIEW

2.1. The Central Line of the Argument

1. **New domains of knowledge δ can be developed by using heuristics.** Radically new concepts and relations connecting them can be discovered by employing a large corpus of heuristics both to suggest plausible actions and to prune implausible ones. To accomplish this requires heuristics of varying levels of generality and power, an adequate representation for knowledge, some initial hypotheses about the nature of domain δ, and the ability to gather data and test conjectures about that domain.

2. **As new domains of knowledge emerge and evolve, new heuristics are needed.** A field may change by the introduction of some new device, theory, technique, paradigm, or observable phenomenon; each time it does so, the corpus of heuristics useful for dealing with that field may also change. Consider the body of heuristics useful in planning a trip from San Francisco to Bern. Over the last century, many new ones have been added, and many old ones have undergone revision.

3. **New heuristics can be developed by using heuristics.** The first two points imply that new heuristics must be discovered. How is this done? Since "Heuristics" is a domain of knowledge, like Electronics, or Mathematics, or Travel planning, perhaps all that is necessary is to set δ = Heuristics in (1) That is, let the field of heuristics itself grow via heuristic guidance. To do this would require many types of heuristics (some quite general, some specific to dealing with other heuristics, etc.), an adequate representation for heuristics, and some hypotheses about the nature of heuristics.

4. **As new domains of knowledge emerge and evolve, new representations are needed.** Just as the potency of a fixed body of heuristics decreases as we move into new fields, so too does the potency of whatever scheme is being used to represent knowledge. Representations must evolve as domain knowledge accretes.

5. **New representations can be developed by using heuristics.** Points (1) and (4) imply that new representations for knowledge must be devised from time to time, and that existing schemes must change. How can this happen? Since "Representation of knowledge" is a field, just as is Mathematics, or Electronics, or Heuristics, or Travel planning, perhaps we can somehow set δ = Representation in (1). That is, allow heuristics to manage the development of new representations.

The final point is that there is no sixth point to make. The preceding five statements comprise a research programme to follow, one plan of attack upon the central problem, the bottleneck of

automatic knowledge acquisition.

Other directions of attack are promising, and are being pursued vigorously by several AI researchers. For most fields, some necessary component required by (1) above is missing (e.g., the automatic acquisition of data is awkward or impossible). In such cases, the human expert must be preserved "in the loop" of Figure 1. Any aids for interviewing the expert are then quite important, tools which facilitate the *manual* knowledge acquisition process depicted in Figure 1. Indeed, much recent AI activity focuses on developing such tools: AGE, EMYCIN, EXPERT, HEARSAYIII, RLL, ROGET, ROSIE, and the various knowledge representation languages.

This paper presents work to date, by the author, along the research programme outlined in Figure 2. Although the development parallels the ordering given therein, the amount of space devoted to each point is not uniform. Much of the paper is concerned with recounting the experience of building AM, a computer program which searches for interesting new concepts and conjectures in elementary mathematics (point (1); see Figure 2 below). The analysis of AM's eventual demise provides an illustration of (2). Much of the remainder is used to develop the rudiments of a theory of heuristics, which theory is required for (3). The paper closes with a detailed example illustrating (3), (4), and (5).

---

(1) New domains of knowledge δ can be developed by using heuristics.
(2) As new domains of knowledge emerge and evolve, new heuristics are needed.
(3) New heuristics can be developed by using heuristics.
(4) As new domains of knowledge emerge and evolve, new representations are needed.
(5) New representations can be developed by using heuristics.

Figure 2: Automatic knowledge acquisition via discovery

2.2. *Controlling the Use of Heuristic Knowledge*

There is an implied "control structure" for the processes of using and acquiring knowledge (solving and proposing problems, using and discovering heuristics, choosing and changing representations, etc.) In fact, it's a nontrivial assumption that a *single* control loop is powerful enough to manage both types of processes. Our experiences with expert systems in the past [Feigenbaum 77] have taught us that the power lies in the knowledge, *not* in the inference engine.

What is that topmost control loop? It assumes that there is a large corpus of heuristics for choosing (and shifting between) representations. From time to time, some of these heuristics evaluate how well the current representations are performing (e.g., is there now some operation which is performed very frequently, but which is notoriously slow in the current representation?) At any moment, if the representations used seem to be performing sub-optimally, some attention will be focused on the problem of shifting to other ones, maintaining the same knowledge simultaneously in multiple representations, devising whole new systems of representation, etc. Similarly, we assume there are several heuristics which monitor the adequacy of the existing stock of heuristics, and as need arises formulate (and eventually work on and solve) tasks of the form "Diagonalization is used heavily, but has no heuristics associated with it; try to find some new specific heuristics for dealing with Diagonalization". A typical rule for working on such a task might say "To find heuristics specific to C, try to analogize heuristics specific to concepts which were discovered the same way that C was discovered".

It is assumed that these representation heuristics and heuristic heuristics have run for a while, and the system is in a kind of equilibrium. The representations employed are well suited to the tasks being performed, and the heuristics being followed serve as quite effective guides for "plausible

move generation" and "implausible move elimination." The system now proceeds for a while along its object-level pursuits , whatever they may be (proving theorems in plane geometry, discovering new concepts in programming, etc.) Gradually, the object level may evolve: new concepts will be uncovered and focused upon, new laboratory techniques will be discovered, long-standing open questions will be answered, etc. As this occurs, the old representations for knowledge, and the old set of guiding heuristics, may become less ideal, less effective. This in turn would be detected by some of the "meta"-heuristics discussed in the last paragraph, and they would cause the system to recover its equilibrium, to search for new representations and new heuristics to deal effectively once again with the objects and operations at the object level.

In other words, new concepts, conjectures, theorems, etc. emerge all the time; as they are investigated, some turn out to be useful and some turn out to be dead-ends; using a fixed set of guiding heuristics, the rate at which useful new discoveries are made will decline gradually over time; eventually it's worth pausing in the search for domain-specific knowledge, and turning instead to the problem of finding new heuristics (perhaps by articulating experiences to date in the task domain). The discoverer later returns to his original task, armed with new and hopefully more powerful heuristics. This cycle of looking for domain concepts, occasionally punctuated by an effort to find new heuristics, continues until, gradually, it becomes harder and harder to find new heuristics. At that point it becomes worthwhile to look for new and different representations for knowledge.

The top-level control structure is thus homeostatic: detecting and correcting for any inappropriateness of representations employed or heuristics employed. For these purposes, we believe it suffices to have (and use) a corpus of heuristics for guidance. Of course that top level loop could itself be implicitly defined by a set of heuristic rules, and we would expect such rules to change from time to time, albeit *very* slowly. If, for example, no new concepts or operations were defined at the object level for a long period of time, then the need for close monitoring of the adequacy of the representations being employed would evaporate. One important point is that it is not necessary to distinguish meta-heuristics from object-level heuristics: they can be represented the same way, they can be managed by the same interpreter, etc. For example, the very general recursive rule "To specialize a complex construct, find the component using the most resources, and replace it by several alternate specializations" applies to specializing laboratory procedures, mathematical functions, heuristics (including itself!), and representational schemes.

3. Heuristics used to develop new knowledge

"How was *X* discovered?" When confronted with such a question, the philosopher or scientist will often retreat behind the mystique of the all-seeing I's: Illumination, Intuition, and Incubation. A different approach would be to provide a rationalization, a scenario in which a researcher proceeds reasonably from one step to the next, and ultimately synthesizes the discovery *X*. In order for the scenario to be convincing, each step the researcher takes must be justified as a plausible one. Such justifications are provided by citing *heuristics*, more or less general rules of thumb, judgmental guides to what is and is not an appropriate action in some situation.

For example, consider the heuristic in figure 3. It says that if a function f takes a pair of A's as arguments, then it's often worth the time and energy to define $g(x)=f(x,x)$, that is, to see what happens when f's arguments coincide. If f is multiplication, this new function turns out to be squaring; if f is addition, g is doubling. If f is union or intersection, g is the identity function; if f is subtraction or exclusive-or, g is identically zero. Thus we see how two useful concepts (squaring, doubling) and four important conjectures might be discovered by a researcher employing this simple heuristic.

IF `f:AxA--> B;`
THEN `define g:A--> B as g(x)=f(x,x)`

Figure 3. A heuristic which leads to useful concepts and conjectures

Elsewhere [Lenat 79], we describe the uses for a heuristic which says "*If f:A---->B, and there is some extremal subset b of B, Then define and study* $f^{-1}(b)$." If f is *Intersection*, this heuristic says it's worth considering pairs of sets which map into extremal kinds of sets. Well, what's an extremal kind of set? Perhaps we already know about extremely small sets, such as the empty set. Then the heuristic would cause us to define the relationship of two sets having empty intersection -- i.e., disjointness. If f is *Employed-as*, then the above heuristic says it's worth defining, naming, and studying the group of people with no jobs (zero is an extremely small number of jobs to hold), the group of people who hold down more than one job (two is an extremely large number of jobs to hold). If f is *Divisors-of*, then the heuristic would suggest defining the set of numbers with no divisors, the set of numbers with one divisor, with two divisors, and with three divisors. The third of these four sets is the concept of prime numbers. Other heuristics cause us to gather data, to do that by dumping each number from 1 to 1000 into the appropriate set(s), to reject the first two sets as too small, to notice that every number in the fourth set is a perfect square, to take their square roots, and finally to notice that they then coincide precisely with the third set of numbers. Now that we have the *definition* of primes, and we have found a surprising *conjecture* involving them, we shall say that we have *discovered* them (note that we are nowhere near a proof of that conjecture).

Of course the above instances of discoveries are really just *reductions*. We can be said to have reduced the problem "How might Squaring be discovered?" to the somewhat simpler problem "How might Multiplication be discovered?" by citing the heuristic in Figure 3. Similarly, we reduced the problem of discovering Primes to the problem of discovering Divisors-of. Such reductions could be continued, reducing the discovery of Divisors-of to that of Multiplication, thence to Addition or Cartesian-product, and so forth. Eventually, we would go down all the way to our conceptual primitives, to concepts so basic that we feel it makes no sense to speak of discovering them. See figure 4.

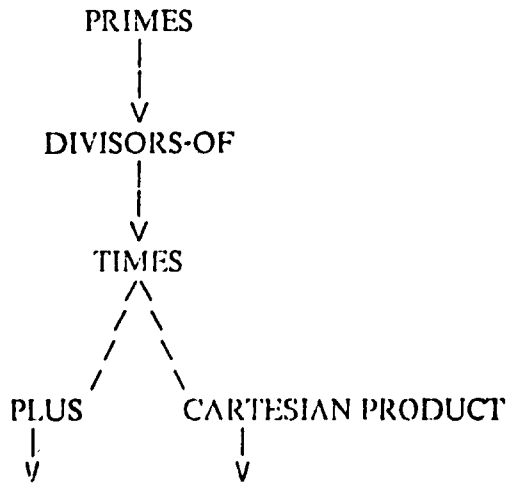

Figure 4. Reducing each concept's discovery to that of a simpler one. Note that multiplication can be discovered if the researcher knows either addition of numbers *or* Cartesian products of sets.

Why, then, is the act of creation so cherished? If some significant discoveries are merely one or two "heuristic applications" away from known concepts, why are even one-step discoveries worth communicating and getting excited about? The answer is that the discoverer is moving upwards in the tree, not downwards. He is not rationalizing, in hindsight, how a given discovery might have been made; rather, he is groping outward into the unknown for some new concept which seems to be useful or interesting. The downward, analytic search is much more constrained than the upward, synthetic one. Discoverers move upwards; axiomatizers, colonizers, and pedagogues move downwards. See Figure 5. Even in this limited situation, the researcher might apply the "Repeat" heuristic to multiplication, and go off along the vector containing exponentiation, hyper-exponentiation, etc. Or he might apply "look at inverse of extrema" to Divisors-of in several ways, for example looking at numbers with *very many* divisors.

Once a discovery has been made, it is much easier to rationalize it in hindsight, to find some path downward from it to known concepts, than it was to make that discovery initially. That is the explanation of the phenomenon we've all experienced after working for a long time on a problem, the feeling *"Why didn't I solve that sooner!"* When the reporter is other than ourselves, the feeling is more like *"I could have done that, that wasn't so difficult!"* It is the phenomenon of wondering how a magic trick ever fooled us, once we've seen the method. It enables us to follow mathematical proofs with a false sense of confidence, being quite unable to prove similar theorems. It is the reason why we can use Polya's heuristics [Polya 45] to parse a discovery, to explain a plausible route to it, yet feel very little guidance from them when faced with a problem and a blank piece of paper.

There still is that profusion of upward arrows to contend with. One of the triumphs of AI has been finding the means to muffle a combinatorial explosion of arrows: one must add some heuristic guidance criteria. That is, add some additional knowledge to indicate which directions are expected to be the most promising ones to follow, in any situation. So by a *heuristic*, from now on, we shall mean a contingent piece of knowledge, such as the top entry in Figure 6, rather than an unconstrained Polya-esque maxim (6b). The former is a heuristic, the latter is an explosive.

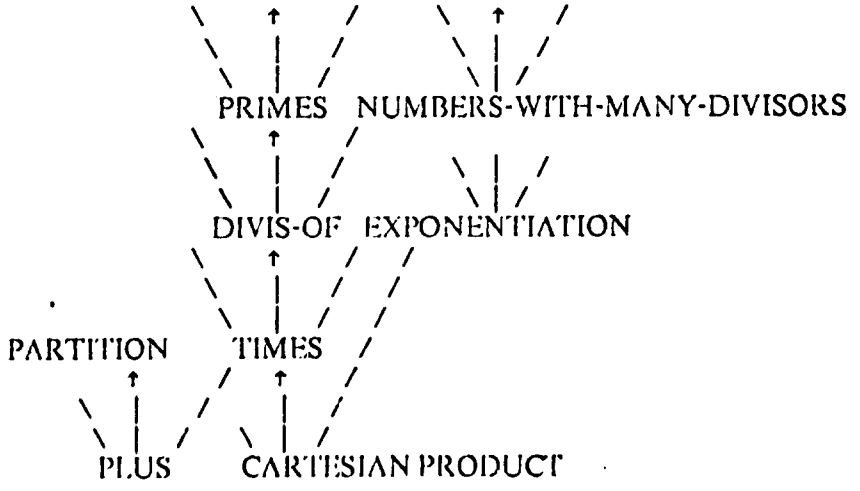

Figure 5. The more explosive upward search for new concepts

(*a*) IF the range of one operation has a large intersection with the domain of a second,
and they both have high worth,
and either there is a conjecture connecting them or
the range of the second operation has a large
intersection with the domain of the first,
THEN compose them and study the result.

(*b*) Compose two operations and study the result.

Figure 6. A contingent heuristic rule and an explosive one.

There is a partial theory of intelligence here, which claims that discovery can be adequately guided by a large collection of such heuristic rules. In particular, mathematical discovery may be so guided. To test this hypothesis, we designed and constructed AM, a LISP program whose task was to explore elementary finite set theory: gathering empirical data, noticing regularities in them, and defining new concepts. AM is well described elsewhere [Lenat 79], and a very brief recapitulation here should suffice.

AM began with one hundred set theory concepts. This included static structures (sets, bags, lists) and many active operations (union, composition, canonize). For each concept, we supplied very little information besides its definition. In addition, AM contained 243 heuristic rules for proposing plausible new concepts, for filling in data about concepts, and for evaluating concepts for "interestingness". Among them are the two heuristics we saw earlier, for looking at the inverse of extrema and for looking at the new function $g(x) =_{df} f(x,x)$.

During the course of its longest run (a couple hours), AM defined several hundred concepts, about half of which were reasonable, and noticed hundreds of simple relationships involving them, most of which were trivial. AM found several set-theoretic concepts (disjointness, de Morgan's laws), defined natural numbers, found arithmetic and elementary divisibility theory, and began to bog down in advanced number theory (after finding the fundamental theorem of arithmetic and Goldbach's conjecture). Each "discovery" involved relying on over 30 heuristics, and almost all heuristics participated in dozens of different discoveries; thus the set of heuristics is not merely "unwound" to produce the discoveries. Since the heuristics did lead to the discoveries, they must in some sense be an encoding for them, but they are not a conscious or (even in hindsight) obvious encoding.

AM's basic control structure was simple: select some slot of some concept, and work to fill in entries for it. Since AM began with over 100 concepts, and each had about 20 slots to fill in (Examples, Generalizations, Conjectures, Analogies, etc.), there were 2000 small tasks for AM to perform, initially. This number grew with time, because new concepts would usually be defined long before 20 slots were filled in on old ones. Each task was placed on an agenda, with symbolic reasons justifying why it should be attended to. Those tasks having several good reasons would eventually percolate to the top of the agenda and be worked on. To accomplish the selected task, AM located relevant heuristics and obeyed them. They in turn caused entries to be filled in on hitherto blank slots, defined entirely new concepts, and proposed new tasks to be added to the agenda.

There is one more issue about AM that should be discussed in this paper: how it was able to efficiently restrict its attention to a small set of potentially relevant heuristics at all times. Consider for a moment the AM heuristic that says "IF a composition fog preserves most of the properties that f had, THEN it's more interesting." That's useful when evaluating the worth of a composition, but of course is of no help when trying to find examples of Sets. We associated that heuristic with the

Composition concept, the most general concept for which it was relevant. Another heuristic AM has says "IF the domain and range of an operation coincide, THEN it's more interesting." That one was tacked onto the Operation concept. But note that since Compositions are special kinds of Operations, the heuristic should apply to them as well. The general principle at work here is the following: *If a heuristic is relevant to C, then it's also relevant to all specializations of C.* If we look at the AM representation for Composition, we would see a frame-like data structure (schema, property list) one of whose slots is Generalizations, and one of the entries therein is Operation. This is AM's way of recording the fact that Composition is a specialization of Operation. The obvious algorithm, then, when dealing with some specific concept C, is to follow Generalization links upward, gathering heuristics tacked onto any concept encountered along the way. See Figure 7. In general, this means that AM's attention is restricted to log(n) heuristics, rather than n. AM can completely ignore all the rest, and need only evaluate the IF parts of these log(n) potentially relevant ones. In other words, the Generalization/Specialization hierarchy of concepts has *induced* a similar powerful structuring upon the set of heuristics. The power of this technique is dimmed somewhat by the unequal distribution of heuristics in the Generalization/Specialization tree: a large number of heuristics clustered near the few topmost (very general) concepts.

As AM forayed into number theory, it had only heuristics from set theory to guide it. For instance, when dealing with prime pairs (twin primes), there were no specific heuristics relevant to them; they were defined in terms of primes, which were defined in terms of divisors-of, which was defined in terms of multiplication, which was defined in terms of addition, which was defined in terms of set-union, which (finally!) had a few attached heuristics. Because it lacked number-theory heuristics, embodying what we would call common-sense about arithmetic, AM's fraction of useless definitions went way up (Numbers which are both odd and even; Prime triples; The conjecture that there is only one prime triple (3,5,7) but without understanding why; etc.) It was unexpected and gratifying that AM should discover numbers and arithmetic at all, but it was disappointing to see the program begin to thrash. When a few dozen concepts from plane geometry were added to AM, the same type of thrashing soon occurred; only the addition of specific geometry heuristics would prevent this collapse.

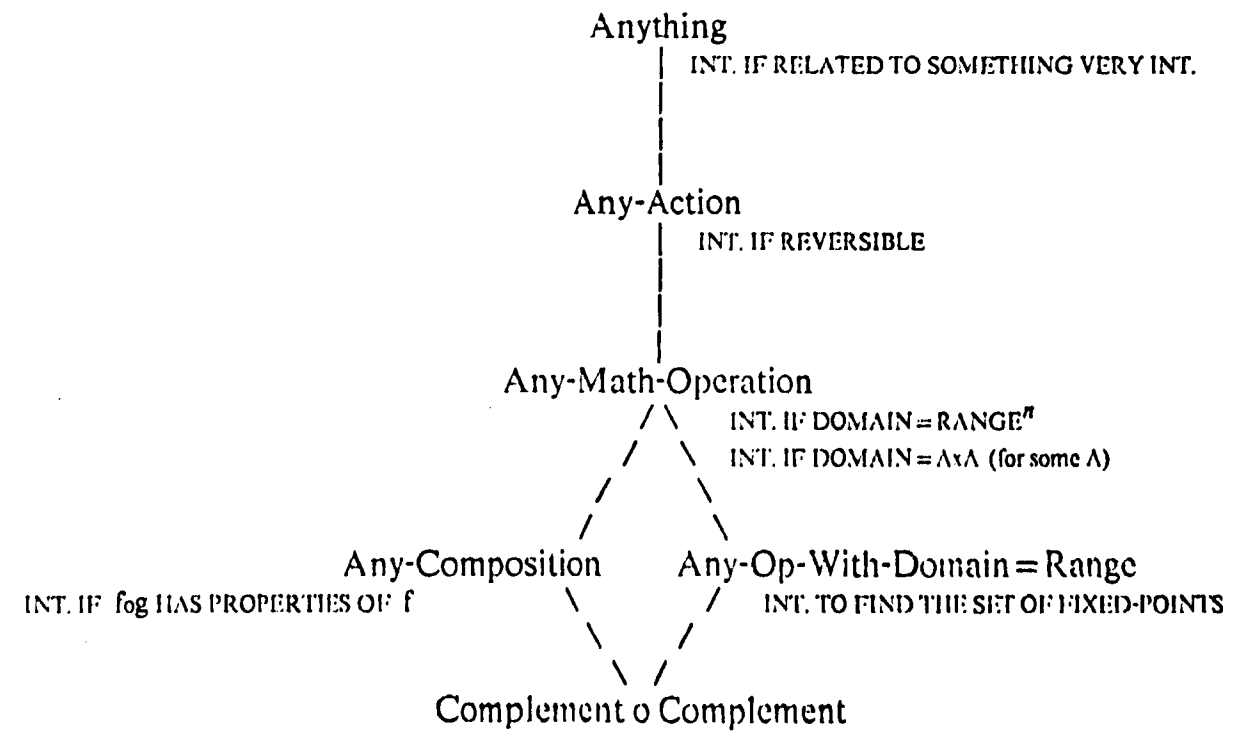

Figure 7. One branch of the Generalization hierarchy of concepts, with a few of the attached heuristics

There are two relevant conclusions from the AM research: (i) It is possible for a body of heuristics to effectively guide a program in searching for new concepts and conjectures involving them. (ii) As new domains of knowledge emerge, the old corpus of heuristics may not be adequate to serve as a guide in those new domains; rather, new specific heuristics are necessary. Notice that these are also the first two points in the argument of this paper (see Figure 2).

Before embarking on point (3) of the central argument of Figure 2, it is necessary to have a "theory of heuristics". Toward that end, we can begin collecting elements of that theory based on our experiences with AM. See Figure 8. One remark, besides the two mentioned in the last paragraph, is that heuristics can be used both to generate promising actions and to prune away poor ones. Thus, AM's search space is never explicitly described; there is no clear notion of a set of legal operators which defines some immense space of syntactic mathematical concepts and conjectures, etc. Any such attempt would probably produce a search space of such size as to be useless (100^{20} in AM's domain of elementary finite set theory). Rather, AM's set of heuristics *implicitly* defines its search space. If you remove a heuristic from AM, it has *less* to do; this is exactly the opposite of the case with most heuristic search programs, where heuristics are used exclusively to prune away implausible paths.

The final remark noted in figure 8 is that the heuristics fall into a nice hierarchy, induced by the one between domain concepts. The key point here is that each heuristic has a domain of relevance: the most general concept to which it's relevant and all the specializations of that concept. This organization enables the interpreter, through simple inheritance, to focus on the *log* of the number of all heuristics in the system, rather than that entire set, at each moment.

(*i*) A SET OF HEURISTICS CAN GUIDE CONCEPT DISCOVERY

(*ii*) A NEW FIELD WILL DEVELOP SLOWLY IF NO SPECIFIC NEW HEURISTICS FOR IT ARE CONCOMITTANTLY DEVELOPED

(*iii*) HEURISTICS CAN BE USED AS PLAUSIBLE MOVE GENERATORS OR AS IMPLAUSIBLE MOVE ELIMINATORS

(*iv*) THE GENERALIZATION/SPECIALIZATION HIERARCHY OF CONCEPTS INDUCES A SIMILAR STRUCTURE UPON THE SET OF HEURISTICS

Figure 8. Elements of a theory of heuristics, learned from work on AM

---

4. Heuristics change as task domains do

Let's continue to explore the notion of a heuristic having a domain of relevance. Consider the following very special situation: you are asked to guess whether a conjecture is true or false. What heuristics are useful in guiding you to a decision rapidly? If the conjecture is in the field of plane geometry, one very powerful technique is to draw a diagram and see whether it holds in that analogic model. But if the conjecture is in the field of point-set topology, or real analysis, this is a *terrible* heuristic which will often lead you into error. For instance, if the conjecture mentions a

function, then any diagram you draw will probably picture a function which is everywhere infinitely differentiable, even if such is never stated in the conjectures's premises. As a result, many properties will hold in your diagram that can never be proven from the conjecture's premises. The appropriate technique in topology or analysis is to pull out your book of 101 favorite counterexamples, and see whether any of them violate the conjecture. If it passes all of them, then you may guess it's probably true.

This example dramatizes the idea that the power or utility of a heuristic changes from domain to domain. Thus, as we move from one domain to another, the set of heuristics which we should use for guidance changes. Many of them have higher or lower utility, some entirely new heuristics may exist, and some of the old ones may be actually *detrimental* if followed in the new domain. For instance, the "IF falling object THEN catch it" rule is useful for most situations, but people are burned when they try to catch falling clothes irons and soldering irons.

Heuristics are compiled hindsight: they are nuggets of wisdom which, if only we'd had them sooner, would have led us to our present state much faster. Even the synthesis of a new discovery via analogy, aesthetic criteria such as symmetry, or random combination, can be considered to be the result of employing guidance heuristics (e.g., "Analogies are useful in formulating biological and sociological theories", "Symmetry is useful in postulating the existence of fundamental particles in physics", "Randomly looking for regularities in elementary number theory and plane geometry may be profitable".) Those guidance heuristics were in turn based on several past episodes, hence are themselves compiled hindsight. Nilsson and others have argued for the primacy of search; we are simply stating the corollory for the very special case where one must let time flow event nodes past us for our observation and recording: the primacy of compiled experiential knowledge.

As new empirical evidence accumulates, it may be useful to recompile the heuristics. Certainly by the time you've opened up a whole new field, you *must* recompile them. Working in point-set topology with geometry heuristics is not very efficient, nor was AM's working in number theory using only heuristics from set theory. The set of heuristics must evolve as well: some old ones are no longer useful, some must be refined to suit the new domain, and some entirely new heuristics may be useful. As the task varies, or as time varies and one gains new experiences, one's set of guiding heuristics is no longer optimal. The utility of a heuristic will vary, then, both across tasks and across time, and this variance is not necessarily continuous.

Exactly what kinds of changes can occur in a domain of knowledge that might require you to alter your set of heuristics? In other words, what are the sources of *granularity* in the space of "fields of knowledge"?

First, there might be the invention of a new piece of apparatus. This could be theoretical (such as Godel's theorem) or technological (such as the computer). Heuristics spring into being: rules which tell you how to use such a thing, when it's relevant, how to fix one, what kind to buy, etc. In addition, many of the old heuristics may be less or (rarely) more useful than they used to be.

Second, there might be a new technique devised, one which doesn't actually depend upon any new apparatus. Again, this can be theoretical (such as Bentley's widespread application of divide and conquer in complexity) or practical (such as Maxam and Gilbert's clever method for sequencing DNA). New heuristics about reliability, applicability, etc. are created, and old ones fade away.

Third, a new phenomenon may be observed. Whenever a new invention occurs, there are often two immediate new phenomena: the sociological one of how the invention is used, and the "real" one now observable using the invention.

Fourth, and most unusually, there may be a newly-explicated or newly-isolated concept or field, one which was always around but never spoken about explicitly. The notion of paradigms is such a concept, and the whole field of heuristics itself is such a field. For example, there exist heuristics for when to apply heuristics, for whom to invite to talk about heuristics, for how to evaluate a heuristic's worth, etc.

In other words, "Heuristics" itself is a field of study. As an analogy, consider the field of

"Grammars". It may be discussed theoretically, independent of any particular language, yet to *develop* that theory the researcher no doubt was always grounded in a context of some language or other. Similarly, to develop a general theory of heuristics one must constantly deal with heuristics for some specific field or task. Eventually the theory of Grammars advanced to the stage of formalization where it no longer needed such grounding, but Heuristics is far from there yet.

In brief, the sources of granularity in the space of "domains of knowledge" are precisely those components which, if varied, lead to a new domain of knowledge. In other words, they *define* what we mean by a domain of knowledge: a set of phenomena to study, a body of specific problems about those phenomena which are considered worth working on, and a set of methods (both theoretical and experimental, mental and material) for attacking such questions. The definition corresponds closely to what Thomas Kuhn has called "paradigms".

This section has now contributed three new elements to our growing theory of heuristics:

(*v*) HEURISTICS ARE COMPILED HINDSIGHT

(*vi*) THE SPACE OF "DOMAINS OF KNOWLEDGE" IS GRANULAR

(*vii*) "HEURISTICS" IS ITSELF A SEPARATE FIELD

Figure 9. Three additional elements of a theory of heuristics

---

5. A Theory of Heuristics

5.1 Why Heuristics Work

The seven items mentioned in Figures 8 and 9 as "elements of a theory of heuristics" actually sound more like 2^{nd}-order correction terms to some as-yet unstated more fundamental theory. What is that basic 0^{th}-order theory? What is the central assumption underlying heuristics? It appears to be the following: "Appropriateness(action,situation) is cts." That is, Appropriateness, viewed as a function of actions and of situations, is a *continuous* function of both variables.

Corollary 1: For a given action, its appropriateness is a continuous function of situation. Heuristics specify which actions are appropriate (or inappropriate) in a given situation. One corollary of the central assumption is that if the situation changes only slightly, then the judgment of which actions are appropriate also changes only slightly. Thus compiled hindsight is useful, because even though the world changes, what was useful in situation *X* will be useful again sometime in situations similar to *X*. There are two special cases of the Corollary 1 worth mentioning: see Figure 10.

0^{th} : Appropriateness(action,situation) is a *continuous* function.

COR. 1: If action A is appropriate in situation S,
Then A is appropriate in most situations which are very similar to S.

COR. 1a: Features of the task environment (task) is continuous.
COR. 1b: World (time) is continuous.

COR 2: If action A is appropriate in situation S,
Then so are most actions which are very similar to A.

Figure 10. The central assumption underlying heuristics, and two special cases

The first of these, Cor. 1a, says that if the task appears to be similar to one you've seen elsewhere, then many of the features of the *task environment* will probably be very similar as well: i.e., the kinds of conjectures which might be found, the solvability and difficulty anticipated with a task, the kinds of blind alleys which one might be trapped in, etc. may all be the same as they were in that earlier case. For instance, suppose that a certain theorem, UFT, was useful in proving a result in number theory. Now another task appears, again proving some number theory result. Because the tasks are similar, Cor. 1a suggests that UFT be used to try to prove this new result. This is the basic justification for using analogy as a reasoning mechanism. A sentiment similar to this was voiced by Poincare' during the last century: *The whole idea of analogy is that 'Effects', viewed as a function of situation, is a continuous function.*

The second special case, Cor. 1b, says that the world doesn't change much over time, and is the foundation for the utility of *memory*. In a world changing radically enough, rapidly enough, memory would be a useless frill; consider the plight of an individual atom in a gas.

Corollary 2: For a given situation, appropriateness is a continuous function of actions. This means that if a particular action was very useful (or harmful) in some situation, it's likely that any very similar action would have had similar consequences. Cor. 2 justifies the use of inexact reasoning, of allocating resources toward finding an approximate answer, of satisficing. It is the basis for employing "generalization" as a mechanism for coping with the world: if the appropriateness function were not (usually) continuous as a function of actions, then most generalizations would be false. One may restate this corollary as "World (situation) is continuous."

If the central assumption holds, then the ideal interpreter for heuristics is the one shown in figure 11. Note that this is very similar to a pure production system interpreter. In any given situation, some rules will be expected to be relevant (because they were truly relevant in situations very similar to the present one). One or more of them are chosen and applied (obeyed, evaluated, executed, fired, etc.) This action will change the situation, and the cycle begins anew. Of course one can replace the "locate relevant heuristics" subtask by a copy of this whole diagram: that is, it can be performed under the guidance of a body of heuristics specially suited to the task of finding heuristics. Similarly, the task of selecting which rule(s) to fire, and in what order, and with how much of each resource available, can also be implemented as an entire heuristic rule system procedure.

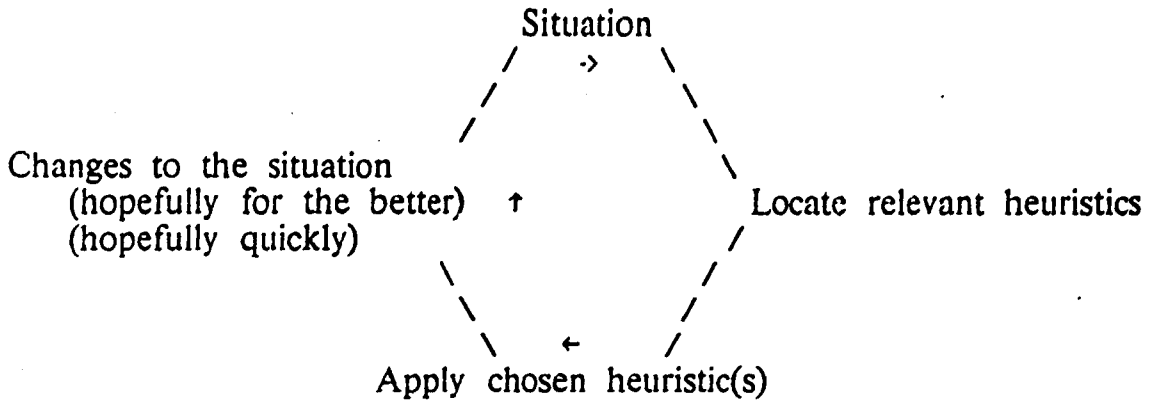

Figure 11. The 0^{th}-order interpreter for a body of heuristic rules

By examining the loop in Figure 11, we can quickly "read off" the possible bugs in heuristics, the list of ways in which a heuristic can be "bad":

It might not be interpretable at all.
It might be interpretable but it might never even be potentially relevant.
It might be potentially relevant but its IF part might never be satisfied.
It might trigger, but never be the rule actually selected for execution (firing).
It might fire, but its THEN part might not produce any effect on the situation.
It might produce a bad effect on the situation.
It might produce a good effect, but take so long that it's not cost-effective.

This is reminiscent of John Seely Brown's work on a generative theory of bugs [Brown & VanLehn 80], and is meant to be. Perhaps by viewing heuristics as performers, this approach can lead to an effective method for diagnosing buggy heuristics, hence improving or eliminating them.

There are several things wrong with the 0th order theory: it presumes that knowledge is complete and unchanging; that is, it ignores the "potato in the tailpipe" problem and the frame problem. The reader may have noticed that the first of the two corollaries in Figure 10 is almost precisely the negation of the empirically-derived statement (*vi*) in Figure 9. The latter claims that the space of task domains is inherently and profoundly quantized; the corollary claims it's continuous. As we said earlier, the items in Figures 8 and 9 are 2^{nd}-order correction terms to a theory of heuristics, and Figure 10 is a very simplified 0^{th}-order theory. Intermediate between them lies a theory which interfaces to each.

That 1^{st}-order theory says that the 0th-order theory is often a very useful fiction. It is cost-effective to behave as if it were true, if you are in a situation where your state of knowledge is very incomplete, where there is nevertheless a great quantity of knowledge already known, where the task is very complex, etc. At an earlier stage, there may have been too little known to express very many heuristics; much later, the environment may be well enough understood to be algorithmized; in between, heuristic search is a useful paradigm. Predicting eclipses has passed into this final stage of algorithmization; medical diagnosis is in the middle stage where heuristics are useful; building programs to search for new representations of knowledge is still pre-heuristic.

1st : IF you are in a complex, knowledge-rich, incompletely-understood world, THEN it is frequently useful to behave as though it were true that appropriateness(action,situation) is continuous in both variables.

Figure 12. The first-order theory of heuristics: the 0^{th}-order theory is a useful fiction.

Notice that the 1^{st}-order theory is itself a heuristic! This is not too disturbing, since it is dubious that we will ever know enough about thinking to supplant it. Until your model of me is *absolutely* perfect, your predictions of my behavior will diverge more and more as time proceeds, and after a relatively short interval you will have to rely upon heuristics again to understand and predict my thoughts and actions. And there is probably something akin to Heisenberg's uncertainty principle to *guarantee* that your model of me can never be perfectly complete.

The second-order corrections in Figures 8 and 9 now apply to the first-order theory, and in addition some new second-order ones are apparent. For instance, the adjective "frequently", used in Figure 12, can be replaced by a body of rules which govern when it is and is not useful to behave so.

5.2. The Power of Each Individual Heuristic

We have discussed the nature of using a corpus of heuristics, but what is the nature of a single one? We've already said that it has some domain of relevance. What does that mean? If we graph the utility or power of the heuristic, as function of task domain, we would expect a curve resembling that of Figure 13. Namely, there is some range of tasks for which the heuristic has positive value. Outside of this, it is often counterproductive to use the heuristic (although the utility may drop to zero rather than falling below zero as pictured). For tasks sufficiently far away, the utility approaches zero, because the heuristic is never even considered potentially relevant, hence never fires. As one example, consider the heuristic that says "If you want to test a conjecture, Then draw a diagram". As we've seen, this has high utility within Euclidean plane geometry, but as the axioms of the theory are changed, its worth declines. By the time you reach point-set topology or real analysis, its value is negative. Eventually, domains like philosophy are reached, where drawing diagrams can rarely be done meaningfully. (As Figs. 13-15 indicate, we hope that "draw a diagram" is a good heuristic for the domain of Heuristics.) As another example, consider the heuristic "If a predicate rarely returns True, Then define new generalizations of it". This is useful in set theory, worse than useless in number theory, and useless in domains where "predicate" is undefined.

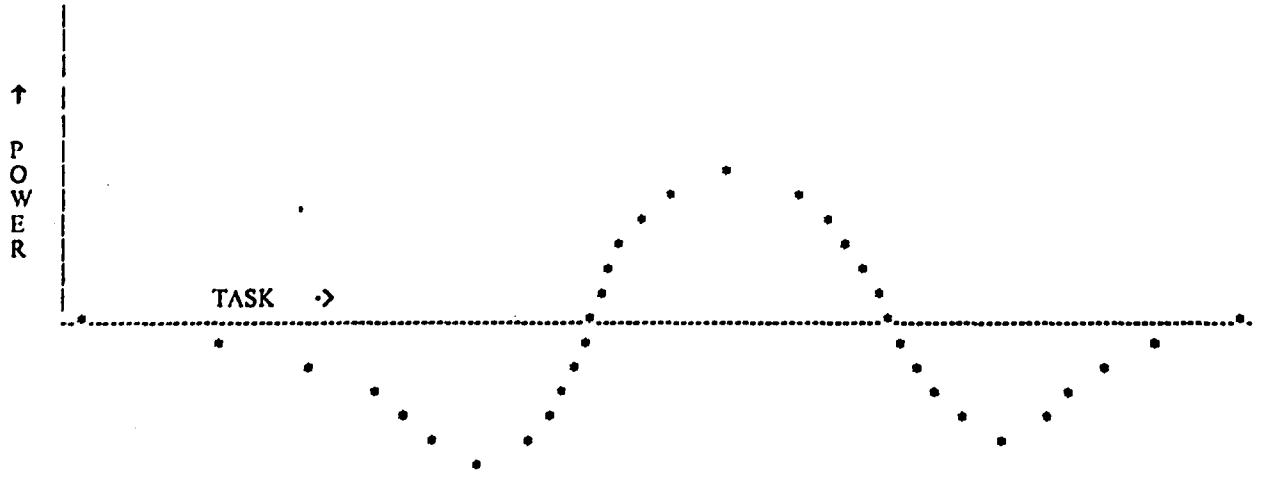

Figure 13. The graph of a heuristic's power, as a function of the task it is applied to.

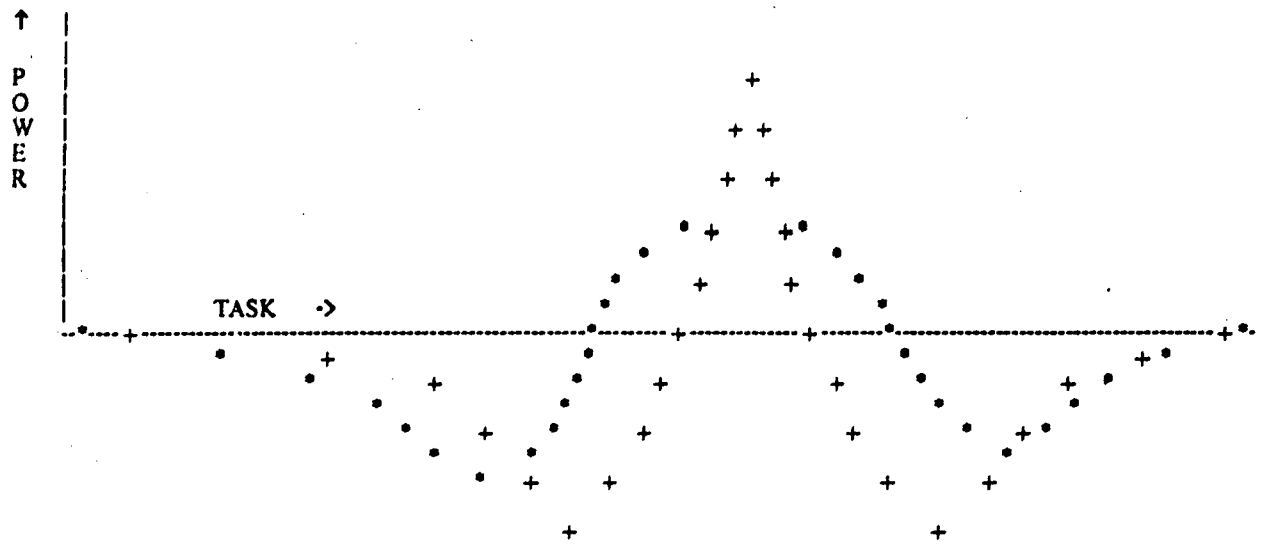

Figure 14. The change in power when a heuristic (*) has its THEN- part specialized (+)

If we specialize the THEN-part of a heuristic, it will typically have higher utility but only be relevant over a narrower domain. See Figure 14. Notice the area under the curve appears to be remain roughly constant; this is a geometric interpretation of the tradeoff between generality and power of heuristic rules. It is also worth noticing that the new specialized heuristic may have negative utility in regions where the old general one was still positive, and it will be meaningless over a larger region as well. Consider for example the case where "Generalize a predicate" is specialized into "Generalize a predicate by eliminating one conjunct from its definition". The latter is more powerful, but only applies to predicates defined conjunctively.

By examining Figure 14, it is possible to generate a list of possible bugs that may occur when the actions (THEN-part) of a heuristic are specialized. First, the domain of the new one may be so narrow that it is merely a spike, a delta function. This is what happens when a general heuristic is replaced by a table of specific values. Another bug is if the domain is not narrowed at all; in such a case, one of the heuristics is probably completely dominated by the other. A third type of bug appears when the new heuristic has no greater power than the old one did. For example, "Smack a vu-graph projector if it makes noise" has much narrower domain, but no higher utility, than the more general heuristic "Smack a device if it's acting up". Thus, the area under the curve is greatly diminished.

While the last paragraph warned of some extreme bad cases of specializing the THEN- part of a heuristic, there are some extreme good cases which frequently occur. The utility (power) axis may have some absolute desirable point along it (e.g., some guarantee of correctness, or optimal efficiency), and by specializing the heuristic it may exceed that threshhold (albeit over a narrow range of tasks). In such a case, the way we *qualitatively* value that heuristic may alter, and indeed we may term it a method, or an algorithm. One way to rephrase this is to say that algorithms are merely heuristics which are so powerful that guarantees can be made about their use. Conversely, one can try to apply an algorithm outside its region of applicability, in which case the result may be useful and that algorithm is then being used as a heuristic. The latter is frequently done in mathematics (e.g., pretending one can differentiate power series expansions to guess at the value of the series). Finally, note that the specialization of the heuristic to one which applies only on a set of measure zero is not necessarily a bad thing: tables of values *do* have their uses.

Specializing the IF-part of a heuristic rule results in its having a smaller region of non-zero utility. That is, it triggers less frequently. As figure 15 shows, this is like placing a filter or window along the x-axis, outside of which the power curve will be absolutely zero. In the best of cases, this

removes the negative-utility regions of the curve, and leaves the positive regions untouched. For example, we might preface the "Draw a diagram" heuristic with a new premise clause, "If you are asked to test a geometry conjecture". This will cause us to use the rule in Geometry situations, where it has been found to have a high utility.

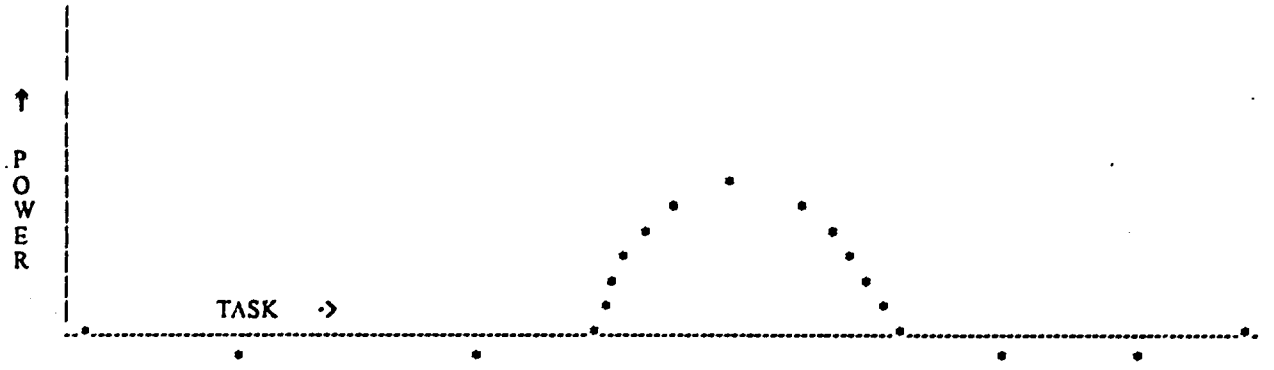

Figure 15. The graph of a heuristic's power, after its IF- part has been optimally specialized.

By examining Figure 15, we can generate a list of possible bugs arising from specializing the conditions (IF-part) of a heuristic rule. The new window may be narrowed to a spike, thus preventing the rule from almost ever firing. There may be no narrowing whatsoever: in that case, it typically would add a little to the time required to test the IF-part of the rule, while not raising the power at all. Of course the most serious error is if it clips away some -- or all! -- of the positive region. Thus, we would not want to replace a general diagram-drawing recommendation with one which advised us to do so only for real analysis conjectures.

The space of domains is granular, quantized, hence these power curves are step-functions (or histograms) rather than smooth curves as we've drawn them. One implication of this is that there is a very precise point along the task axis where the utility drops from positive to negative (or zero). Often, this is a very large, very sudden drop across a single discontinuity in the axis (e.g., when a product emerges, an expert dies, a theorem is proved.)

What are implications of this simple "theory of heuristics"? One effect is to determine in what order heuristics should be chosen for execution; this is discussed in the next paragraph. A second effect is to indicate some very useful slots that each heuristic can and should have, attributes of a heuristic that can be of crucial importance: the peak power of the rule, its average power, the sizes of the positive and negative regions (both projections along the task axis (x-axis) and the areas under the curves), the steepness with which the power curve approaches the x-axis, etc. Let us take the last attribute to illustrate. Why is it useful to know how steeply the power curve approaches Utility = 0 (the x-axis)? If this is *very* steep, then it is worth investing a great amount of resources determining whether the rule is truly relevant in any situation (for if it is slightly irrelevant, then it may have a huge negative effect if used). Conversely, if the slope is very gentle, then very little harm will result from slightly-inappropriate applications of the rule, hence not much time need ever be spent worrying about whether or not it's truly relevant to the situation at hand.

The whole process of drawing the power curves for heuristics is still conjectural. While a few such graphs have been sketched, there is no algorithm for plotting them, no library of thousands of catalogued and plotted heuristics, not even any agreement on what the various power and task axes should be. Nevertheless, it has already proven to be a useful metaphor, and has suggested some important properties of heuristics which should be estimated (such as the just-mentioned downside risk of applying a heuristic in a slightly inappropriate situation). It is a qualitative, empirical-theory [Newell & Simon 76], and predicts the form that a quantitative theory might assume.

How should heuristics be chosen for execution? In any given situation, we will be at a point along the x-axis, and can draw a vertical line (in case of multi-dimensional task axes, we can imagine a hyperplane). Any heuristics which have positive power (utility) along that line are then useful ones to apply (according to our theory of heuristics), and the ones with high power should be applied before the ones with low power. Of course, it is unlikely we would know the power of a heuristic precisely, in each possible situation; while diagrams such as Figs. 13-15 may be suggestive, the data almost never is available to draw them quantitatively for a given heuristic. It is more likely that we would have some measure of the average power of each heuristic, and would use that as a guess of how useful each one would be in the current situation. Since there is a tradeoff between generality and power, a gross simplification of the preceding strategy is simply to apply the most specific heuristic first, and so on. This is the scheme AM used, with very few serious problems. If all heuristics had precisely the same multiple integral of their power curves, this would coincide with the previous scheme. Of course, there are always some heuristics which, while being very general, really are the most important ones to listen to if they ever trigger ("If a conflagration breaks out, Then escape it").

Notice that the "generality vs. power" tradeoff has turned into a statement about the conservation of volumes in nxm-dimensional space, when one takes the multiple integral of all the power curves of a heuristic. In particular, there are tradeoffs among all the dimensions: a gain along some utility dimension (say Convincingness) can be paid for by a decrease along another (say Efficiency) or by a decrease along a task dimension (a reduction of breadth of applicability of the heuristic). One historically common bug has been over-reliance upon (and glorification of) heuristics which are pathologically extreme along some dimension: tables, algorithms, weak methods, etc.

Heuristics are often spoken of as if they were incomplete, uncertain knowledge, much like mathematical conjectures or scientific hypotheses. This is not necessarily so. The epistemological status of a heuristic, its justification, can be arbitrarily sound. For example, by analyzing the optimal play of Blackjack, a rather complex table of appropriate actions (as a function of situation) is built up. One can simplify this into a "Basic Strategy" of just a few rules, and know quite precisely just how well those rules should perform. That is, heuristics may be built up from systematic, exhaustive search, from "complete" hindsight. Another example of the formal, complete analysis of heuristic methods is well known from physics, where Newtonian mechanics is known to be only an approximation to the world we inhabit. Relativistic theories quantify that deviation precisely. But rather than supplanting Newtonian physics, they *bolster* its use in everyday situations, where its inadequacies can be quantitatively shown to be too small to make worthwhile the additional computation required to do relativistic calculations.

Many, nay most, heuristics *are* merely conjectural, empirical, aesthetic, or in other ways epistemologically less secure than the Basic Strategy in Blackjack and Newtonian physics. The canonical *use* of heuristics is to pretend they are true; the canonical use of a conjecture is to guide a search for a proof of it. If a conjecture turns out to be false, it may yet stand as a useful heuristic.

5.3 The Space of Heuristics

The utility of an entire *set* of heuristics can be graphed as a function of the tasks it's being applied to, and, not surprisingly, produces a curve similar to the one in Figure 13. Hopefully, the set of heuristics is more useful than any member, thus it is probably much broader and taller (or less negative) than any single heuristic inside it. One cannot simply "add" the curves of its members; the interactions among heuristics are often quite strong, and independence is the exception rather than the rule. Often, two heuristics will be different methods for getting to the same place, or one will be a generalization or isomorph of the other, etc., and as a result the set will really not benefit very much from having both of them present. On the other hand, sometimes heuristics interact synergistically, and the effects can be much *greater* than simple superposition would have predicted. The opposite of this simetimes happens: two experts have given you heuristics which separately work, yet which contradict each other. Using either half-corpus would solve your problem, but mixing them causes chaos (e.g., one mathematician gives you heuristics for finding empirical

examples and generalizing, while a second gives you heuristics for formally axiomatizing the situation; either may suffice, the unstructured mixing of the two sets can be catastrophic).

No treatment of heuristics can be complete without some consideration of the space of all the world's heuristics. Consider arranging them in a generalization/specialization hierarchy, with the most general ones at the top. At that top level lie the so-called weak methods (generate & test, hill-climbing, matching, means-ends analysis, etc.) At the bottom are millions of very specific heuristics, involving domain-specific terms like "King-side" and "DDT". In between are heuristics such as those illustrated in Figure 16. A purely "legal-move" estimate of the size of this tree gives a huge final number: Based on the lengths and vocabularies of heuristic rules in AM, one may suppose that there are about 20 blanks to be filled in in a typical heuristic, and about 100 possible entries for each blank (predicate, argument, action, etc.) related to AM's math world. So there are 10^{40} syntactically well-formed heuristics just in the elementary mathematics corner of the tree. Of course, most of these are never (thankfully!) going to fire, and almost all the rest will perform irrelevant actions when they do fire. From now on, let's restrict our attention to the tree of only those heuristics which have positive utility at least in some domains.

What does that tree actually look like? One can take a specific heuristic and generalize it gradually, in all posible ways, until all the generalizations collapse into weak methods. Such a preliminary analysis led us to expect the tree to be of depth about 50, and in the case of an expert system with a corpus of a thousand rules, we might expect a picture of them arranged so to form an equilateral triangle. But if one draws the power curves for the heuristics, it quickly becomes apparent that most generalizations are no less powerful than the rule(s) beneath them! Thus the specific rule can be eliminated from the tree. The resulting tree has depth of roughly 3 or 4, and is thus incredibly shallow and bushy. Professors Herbert Simon, Woody Bledsoe, and the author analyzed the 243 heuristics from AM, and were able to transform their deep (depth 12) tree into an equivalent one containing less than fifty rules and having depth of only four. Looking at a few heuristics arranged in a tiny tree (Fig. 16), we can see that all but the top and bottom levels can be eliminated. A similar phenomenon was seen earlier, in the case of a heuristic which said to smack a vu-graph projector in case it acted up; it and several levels of its generalizations can be eliminated, since they are no more powerful than the general "Smack a malfunctioning device" heuristic. Some *very* specific rule, such as "Smack a Chinook 807 vu-graph projector on its right side if it hums", might embody some new, powerful, specific knowledge (such as the location of the motor mount and this brand's tendency to misalign), and thus need to stay around.

This "shallow-tree" result should make advocates of weak methods happy, because it means that there really is something special about that top level of the hierarchy. Going even one level down means paying attention not to an additional ten or twenty heuristics, but to hundreds. It should also please the knowledge engineering advocates, since most of the very specific domain-dependent rules also had to remain. It appears, however, to be a severe blow to those of us who wish to automatically synthesize new heuristics via specialization, since the result says that that process is usually going to produce something no more useful than the rule you start with. Henceforth, we shall term this the shallow-tree *problem*.

There are two ways out of this dilemma, however. Notice that "utility of a heuristic" really has several distinct dimensions: efficiency, flexibility, power for pedagogical purposes, usefulness in future specializations and generalizations, etc. Also, "task features" has several dimensions: subject matter, resources allotted (user's time, cpu time, space, etc.), degree of complexity (e.g., consider Knuth's numeric rating of his problems' difficulty), time (i.e., date in history), paradigm, etc. If there are n utility dimensions and m task dimensions, then there are actually nxm different power curves to be drawn for each heuristic. Each of them may resemble the canonical one pictured in Figure 13. If by specializing a heuristic we create one which has the appearance of Figure 14 *in any one of these nxm graphs*, then it is a useful specialization. So, while a specialization is unlikely to be useful in any particular utility/task graph, it is quite likely to be useful according to *some* one of the nxm such graphs.

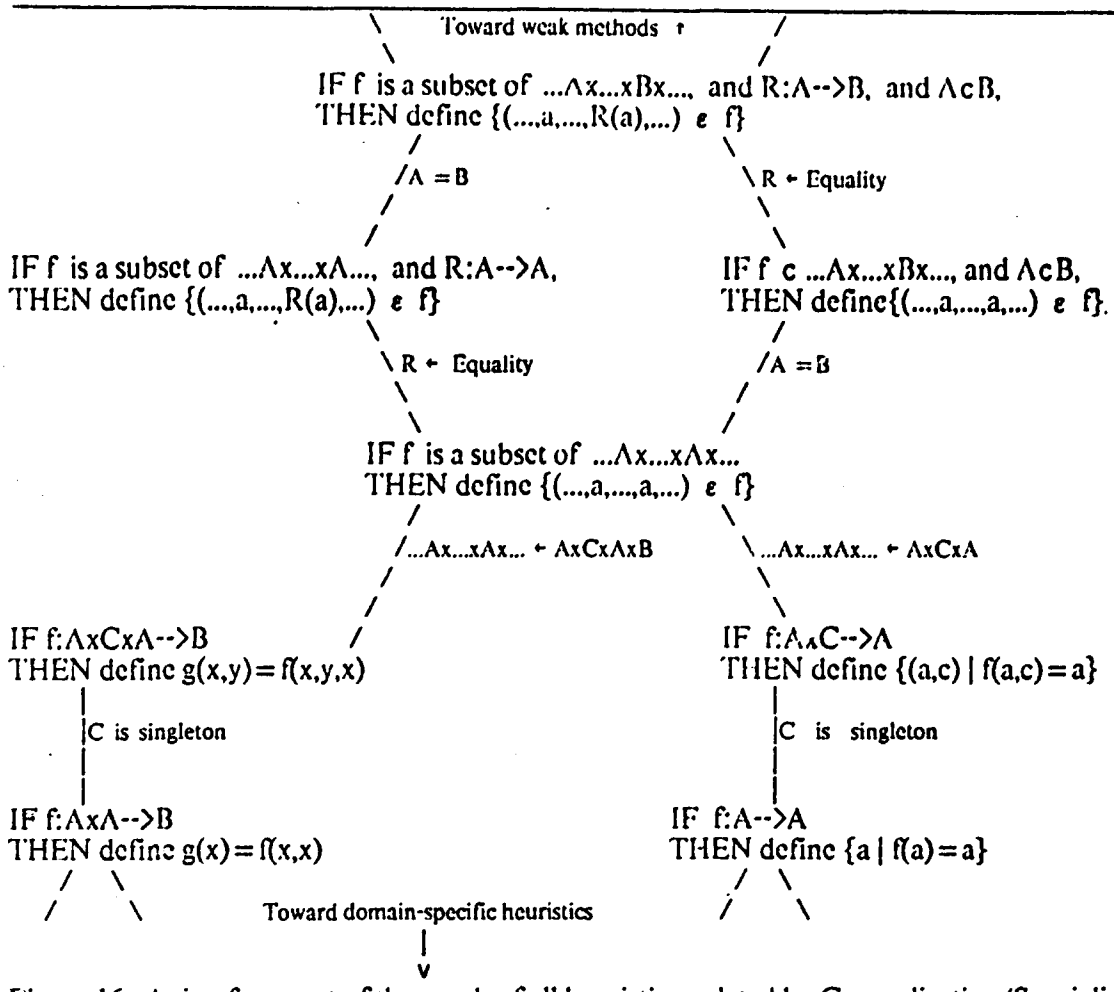

Figure 16. A tiny fragment of the graph of all heuristics, related by Generalization/Specialization. Note the similar derivation of Coalescing and Fixed-Points heuristics.

Consider the Focus of Attention heuristic, that is, one which recommends pursuing a course of action simply because it's been worked on recently. Using this as one reason to support tasks on its agenda made AM *appear* more intelligent to human observers, yet actually take *longer* to make any given discovery. Thus, it is useful in the "Convincingness" dimension of utility, but may be harmful vis a vis "Efficiency".

As another example, consider the heuristics "Smack a vu-graph projector if it's acting up", "Smack a child if it's acting up", and "Smack a vu-graph projector or child if it's acting up". There may be some utility dimensions in which the third of those is best (e.g., scope, humor). However the rationale or justification for the first two heuristics is quite different (random perturbation toward stable state *versus* reinforcement learning). Therefore the third heuristic is probably going to be deficient along other utility dimensions (clarity, usefulness for analogizing).

But there is an even more basic way in which the "shallow tree" problem goes away. There are really a hundred different useful relationships that two heuristics can have connecting them (Possibly-triggers, More-restrictive-IF-part, Faster, My-average-power-higher-than-your-peak-power, Asks-fewer-questions-of-the-user, etc.) For each such relation, an entire graph (note that even the Genl/Spec relation generated a graph, not a tree -- see Fig. 16) can be drawn of all the world's heuristics (or all those in some given program). In some of these trees or graphs, we will find the broad, shallow grouping that was found for the AM heuristics under Genl/Spec. For others, such as Possibly-Triggers, we may find each rule pointing to a small collection of other rules, and hence

the depth would be quite large. There are still many difficult questions to study, even with the theory in this primitive state: e.g., How does the shape of the tree (the graph of heuristics related by some attribute R) relate to the the ways in which R ultimately proves itself to be useful or not useful? Already, one powerful correlation seems to be recognized: In cases where the tree depth is great, that relation is a good one to generalize and specialize along; in cases where the resulting tree is very broad and shallow, other methods (notably analogy) may be more productive ways of getting new heuristics.

6. Heuristics used to develop new heuristics

6.1 Meta-Heuristics are Just Heuristics

Assuming that "Heuristics" is another field of knowledge, just like Electronics or Mathematics, it should be possible to discover new ones and to modify existing ones by employing a large corpus of heuristics. Is there something special about the heuristics which inspect, gather data about, modify, and synthesize other heuristics? That is, should we distinguish "meta-heuristics" from "domain heuristics"? According to our general theory, as presented in the last section, domains of knowledge are granular but nearly continuous along every significant axis (complexity of task, amount of quantification in the task, degree of formalization, etc.) Thus, our first hypothesis should be that it is not necessary to differentiate meta-level heuristics from object-level heuristics -- nay, that it may be artificial and counterproductive to do so.

Figure 17 illustrates two heuristics which can deal with both heuristics and mathematical functions. The first one says that if some concept f has always led to bad results, then f should be marked as less valuable. If a mathematical operation, like Compose, has never led to any good new math concepts, then this heuristic would lower the number stored on the Worth slot of the Compose concept. Similarly, if a heuristic, like the one for drawing diagrams, has never paid off, then *its* Worth slot would be decremented.

The second heuristic says that if some concept has been occasionally useful and frequently worthless, then it's cost-effective to seek new, specialized versions of that concept, because some of them might be much more frequently utile (albeit in narrower domains of relevance). Composition of functions is such a math concept -- it has led AM to some of its biggest successes and failures; this heuristic would add a task to AM's agenda, which said "Find new specializations of Compose". When it was eventually worked on, it could result in the creation of new functions, such as "Composition of a function with itself", "Composition resulting in a function whose domain and range are equal", "Composition of two functions which were derived in the same way", etc. This second heuristic also applies to heuristics, in fact it applies to *itself*. It itself is sometimes useful and sometimes not, and so frequently it truly does pay to seek new, specialized variations of that heuristic. Four possible specializations are, for example, heuristics which demand that f has proven itself useful at least 3 times, that f be specialized in an extreme way, that f have proven itself extraordinarily useful at least once, and that the specializations still be capable of producing any of the successful past creations of f.

IF the results of performing f have always been numerous and worthless,
THEN lower the expected worth of f

IF the results of performing f are only occasionally useful,
THEN consider creating new specializations of f

Figure 17. Two heuristics capable of working on heuristics as well as math concepts

---

6.2 *Attributes of a Heuristic*

In AM, heuristics examine existing frame-like concepts, and lead to new and different concepts. To have heuristics operate on and produce heuristics, it suffices to represent each heuristic as a full-fledged frame-like concept. Thus, the first heuristic listed in Figure 17 needs to reset the value of the Worth slot of the concept f it operates on, and even if f is a heuristic it must have a Worth slot. Similarly, a heuristic that referred to such slots as Average-running-time, Date-created, Is-a-kind-of, Number-of-instances, etc. could only operate upon units (be they mathematical functions or heuristics) having such slots. Figure 18 illustrates (some of the slots from) a heuristic represented in that way. Notice its similarity to the representation of a mathematical operation (Figure 19). The heuristic resembles the function (compare Figs 18-19) much more than the math function resembles the static math concept (compare Figs 19-20).

Earlier, we defined a heuristic to be a contingent piece of guidance knowledge: In some situation, here are some actions that may be especially fruitful, and here are some that may be extremely inappropriate. While some heuristics have pathological formats (e.g., algorithms which lack contingency; delta function spikes which can be succintly represented as tables), most heuristics seem to be naturally stated as rules having the format "IF *conditions*, THEN *actions*." As the body of heuristics grows, the conditions fall into a few common categories (testing whether the rule is potentially relevant, testing whether there are enough available resources to expect the rule to work successfully to completion, etc.) and so do the actions (add new tasks to the agenda, print explanatory messages, define new concepts, etc.) Each of these categories is worth making into a separate named attribute which heuristic rules can possess; Sections 6.3 and 7 will show the power which can arise from drawing such distinctions. So instead of a heuristic having an IF slot and a THEN slot, it will have a bundle of slots which together comprise the conditions of applicability of the heuristic, and another bundle of slots which comprise the actions. See Figure 18. It is also worth defining compound slots in terms of these: a composite IF part, a composite THEN part, a combined IF/THEN lump of LISP code, a compiled version of the same, etc.

All the previous attributes have been *effective*, executable conditions and actions. These are paramount, since they serve to define the heuristic -- they are the *criterial* slots. Many non-effective non-criterial slots are important as well, for describing the heuristic. Some of these relate the heuristic to other heuristics (Generalizations, Specializations, classes of heuristics (Isa), and non-heuristic concepts (View.) Several slots record its origins (Defined-using, Creation-date) and the case studies of its uses so far (Examples).

Once a rich stock of slots (types of attributes) is present for heuristics, several new ones can be derived from them in two ways. First, one can take a slot and ask some questions about it: how does it evolve over time in length, what relationships exist among entries that fill it, how useful are those values, etc. Each such question spawns a new kind of slot (AvgNumberOfExtremeExamples, RelnsAmongMyExtremeExamples, AvgWorthOfExtremeExamples). Second, one can take a pair of slots (say ThenConjecture and If-Truly-Relevant) and a relation (such as Implies) and define a new unary function on heuristics -- a new kind of slot that any heuristic can have -- where H1 would list H2 as an entry on that slot only if (in the present case) the ThenConjecture slot of H1 Implies the IfTrulyRelevant slot of H2. A good name for this new slot might be "CanTrigger", because it lists some heuristics which might trigger when H1 is fired. Of course not all of the n^2 "cross-term" type slots are going to be useful, but this provides a *generator* for a large space of potentially worthwhile new slots. Some heuristics can guide the system in selecting plausible ones to define, monitoring the utility of each selection, and obliterating any which appear empirically rarely to lead to any significant future solutions or discoveries. An example of such a process is given in Section 7.

NAME: Generalize-rare-predicate
ABBREVIATION: GRP
STATEMENT
English: If a predicate is rarely true, Then create generalizations of it
IF-just-finished-a-task-dealing-with: a predicate P \ THESE 3 ATTRIBUTES COMPRISE
IF-about-to-work-on-task-dealing-with: an agenda A |--- IF-POTENTIALLY-RELEVANT
IF-in-the-middle-of-a-task-dealing-with: *never* /
IF-truly-relevant: P returns True less than 5% of Average Predicate
IF-resources-available: at least 10 cpu seconds, at least 300 cells
THEN-add-task-to-agenda: Fill in entries for Generalizations slot of P
THEN-conjecture: P is less interesting than expected
Generalizations of P may be better than P
Specializations of P may be very bad
THEN-modify-slots: Reduce Worth of P by 10%
Reduce Worth of Specializations(P) by 50%
Increase Worth of Generalizations(P) by 20%
THEN-print-to-user: English(GRP) with "a predicate" replaced by P
THEN-define-new-concepts:
CODED-IF-PART: λ(P) ... <LISP function definition omitted here>
CODED-THEN-PART: λ(P) ... <LISP function definition omitted here>
CODED-IF-THEN-PARTS: λ(P) ... <LISP function definition omitted here>
COMPILED-CODED-IF-THEN-PARTS: #30875
SPECIALIZATIONS: Generalize-rare-set-predicate
Boundary-Specializations: Enlarge-domain-of-predicate
GENERALIZATIONS: Modify-predicate, Generalize-concept
Immediate-Generalizations: Generalize-rare-contingent-piece-of-knowledge
Siblings: Generalize-rare-heuristic
IS-A: Heuristic
EXAMPLES:
Good-Examples: Generalize Set-Equality into Same-Length
Bad-Examples: Generalize Set-Equality into Same-First-Element
CONJECTURES: Special cases of this are more powerful than Generalizations
Good-Conjec-Units: Specialize, Generalize
ANALOGIES: Weaken-overconstrained-problem
WORTH: 600
VIEW: Enlarge-structure
ORIGIN: Specialization of Modify-predicate via empirical induction
Defined-using: Specialize
Creation-date: 6/1/78 11:30
HISTORY:

NGoodExamples: 1	NBadExamples: 1
NGoodConjectures: 3	NBadConjectures: 1
NGoodTasks-added: 2	NBadTasksAdded: 0
AvgCpuTime: 9.4 seconds	AvgListCells: 200

Figure 18. Frame-like representation for a heuristic rule from AM. The rule is composed of nothing but attribute:value pairs. After each attribute or slot (often heavily hyphenated) is a colon, and then a list of the entries or values for that attribute of the GRP heuristic.

NAME: Compose
ABBREVIATION: - o -
STATEMENT
English: Compose two functions F and G into a new one FoG
DOMAIN: F, G are functions |--- IF-POTENTIALLY-RELEVANT
IF-truly-relevant: Domain of F and Range of G have some intersection
IF-resources-available: at least 2 cpu seconds, at least 200 cells
THEN-add-task-to-agenda: Fill in entries for some slots of FoG
THEN-conjecture: Properties of F hold for FoG
Properties of G hold for FoG
THEN-modify-slots: Record FoG as an example of Compose
THEN-print-to-user: English(Compose)
THEN-define-new-concepts: Name FoG;
ORIGIN Compose F,G;
WORTH: Average(Worth(F),Worth(G))
DEFN: Append(Defn(G),Defn(F))
Avg-cpu-time: Plus(Avg-cpu(F),Avg-cpu(G))
IF-Potentially-Rele: If-Potentially-Rele(G)
IF-Truly-Rele: If-Truly-Rele(G)
CODED-IF-PART: λ(F,G) ...
CODED-THEN-PART: λ(F,G) ...
CODED-IF-THEN-PARTS: λ(F,G) ...
COMPILED-CODED-IF-THEN-PARTS: #30876
SPECIALIZATIONS: Composition-of-bijections
GENERALIZATIONS: Combine-concepts
Immediate-Generalizations: Combine-functions
IS-A: Function
EXAMPLES:
Good-Examples: Compose Count and Divisors
Bad-Examples: Compose Count and Count
CONJECTURES: Composing F and F is sometimes very good and usually bad
ANALOGIES: Sequence
WORTH: 700
VIEW: Append
ORIGIN: Specialization of Append-concepts with slot = Defn
Defined-Using: Specialize
Creation-date: 11/4/75 03:18
HISTORY:

NGoodExamples: 14	NBadExamples: 19
NGoodConjectures: 2	NBadConjectures: 1
NGoodTasks-added: 57	NBadTasksAdded: 34
AvgCpuTime: 1.4 seconds	AvgListCells: 160

Figure 19. Frame-like representation for a mathematical function from AM.

NAME: Primes
STATEMENT
English: Numbers with two divisors
SPECIALIZATIONS: Odd-primes, Small-primes, Pair-primes
GENERALIZATIONS: Positive numbers
IS-A: Class of numbers
EXAMPLES:
Extreme-exs: 2,3
Extreme-non-exs: 0,1
Typical-exs: 5,7,11,13,17,19
Typical-non-exs: 34, 100
CONJECTURES:
Good-conjecs: Unique-factorization, Formula-for-d(n)
Good-conjec-units: Times, Divisors-of, Exponentiate, Nos-with-3-divis, Squaring
ANALOGIES: Simple Groups
WORTH: 800
ORIGIN: Divisors-of[1] (Doubletons)
Defined-using: Divisors-of
Creation-date: 3/19/76 18:45
HISTORY:
NGoodExamples: 840
NBadExamples: 5000
NGoodConjectures: 3
NBadConjectures: 7

Figure 20. Frame-like representation for a static mathematical concept from AM.

6.3 Discovering a New Heuristic

The AM heuristics create new concepts via specializing existing ones, generalizing (either from existing ones or from newly-gathered data), and analogizing. These are the three "directions" new heuristics will come from. We have exemplified specialization already. One point about generalization is worth making: Heuristics which serve as plausible move generators originate by generalizing from past successes; heuristics which prune away implausible moves originate by generalizing from past failures. Since successes are much less common than failures, it is not surprising that most heuristics in most heuristic search programs are of the pruning variety. In fact, many authors define heuristic to mean nothing more than a pruning aid.

One of the typical "common sense number theory" heuristics which AM lacked was the one which decides that the unique factorization theorem is probably more significant than Goldbach's conjecture, because the first has to do with multiplication and division, while the latter deals with addition and subtraction, and Primes is inherently tied up with the former operations. How could such a heuristic be discovered automatically? This is the starting point for the example we now begin, an example which concludes in the following section, "Heuristics to develop new representations". Why should this be so? That is, what in the world does discovering heuristics have to do with representation of knowledge?

The connection between heuristics and representation is profound. Consider even the special case where we restrict our representations to frame-like ones. The larger the number of different kinds

of slots that are known about, the fewer keystrokes are required to type a given frame (concept, unit) in to the system. Thus, if NGoodConjecs weren't known, it might take 40 keystrokes rather than 1 to assert that there were 3 good conjectures known involving prime numbers. Moreover, no special-purpose machinery to process such an assertion would be known to the system.

This is akin to the power Interlisp derives from the *thickness* of its manual, from the huge number of useful predefined functions. A broad vocabulary streamlines communication. Not only does a profusion of slot types facilitate entering a concept, it makes it easier to modify it once it's entered. Finally, it makes it easier to discover it in the first place; think of it as combining terms in a more powerful, higher level language.

So we see that the task of discovering heuristics should be profoundly accelerated -- or retarded -- by the choice of slots we make for our representation. In the case of an excellent choice of slots, a new heuristic would frequently be simply a new entry on one slot of some concept. Let's see how that can be.

Recall that primes were originally discovered by the system as extrema of the function "Divisors-of". This was recorded by placing the entry "Divisors-of" in the slot called "Defined-using" on the concept called "Primes" (see Figure 20). Later, conjectures involving Primes were found, empiricially-observed patterns connecting Primes with several other concepts, such as Times, Divisors-of, Exponentiation, and Numbers-with-3-divisors. This is recorded on the GoodConjecUnits slot of the Primes concept. Notice that all the entries on Primes' DefinedUsing slot are also entries on its GoodConjecUnits slot. This recurred several times, that is for several concepts besides Primes, and ultimately the heuristic H9 (below) became relevant (its IF-part became satisfied):

H9: IF (for many units *u*) most of the entries on *u.r* are also entries on *u.s*,
THEN-ASSERT that *r* is a subslot of *s* (with justification H9)

This heuristic said that it would probably be productive to pretend that DefinedUsing was always a subslot of GoodConjecUnits. Thus, as soon as you define a new concept X in terms of Y, you should expect there to be some interesting conjectures between X and Y. This new expectation is a new heuristic; in our old, cumbersome IF/THEN language we might express it by two rules saying:

(A) "IF a concept is created with a value in its DefinedUsing slot,
THEN place that value in its GoodConjecUnits slot, with justification H9."

(B) "IF Y is an entry on the GoodConjecUnits slot of X, but no good conjecture between X and Y is yet known, THEN propose a task for the agenda, to look for conjectures between X and Y."

The second of these, (B), has nothing to do with DefinedUsing slots. In fact, it is really no more powerful than a combination of (i) a very general rule that says to verify suspected members of any given slot, and (ii) enough facts about GoodConjecUnits and Conjectures to know how to apply (i) to them. The first one, (A), is the "new heuristic" synthesized by H9. It needn't be represented as shown above; rather, we can simply go to the concept called DefinedUsing (the data structure which holds all the information the program knows about that kind of slot in general), and record that one of its Superslots is GoodConjecUnits. We should also give this an explicit justification, namely H9, since it is a heuristic, not a fact. Figure 21 shows what this record looks like in our current program. The new heuristic is simply the line or two emboldened below; all the non-bold text was present in the program already (though most of it was written by the program itself at earlier times, not filled in by human hands).

It is important to make clear that the semantics of a value *v* appearing as an entry on slot *s* of concept *c* does *not* necessarily mean that it is formally proven that *v* merits a position there; rather, it is merely plausible. Any entry *v* can have an explicit justification, but in lieu of any information to the contrary, the default justification is merely empirical. Thus, when an entry, say Palindromes, is on the GoodConjecUnits slot of Primes, it may mean that some interesting conjectures have been found between Primes and Palindromes, *or* just that it is suspected -- and expected -- that such conjectures can be found if one spends the trouble looking for them.

Thanks to the large number of useful specialized slots, large IF- THEN- rules can be compactly, conveniently, efficiently represented as simple links. Some of these useful slots are very general, but many are domain dependent. Thus, as new domains of knowledge emerge and evolve, new kinds of slots must be devised if this powerful property is to be preserved. The next natural question is, therefore, "How can useful new slots be found?" The last two sentences are the final two points of our original five-point programme (Figure 2), and the next section answers them by way of continuing the example we've begun in this section.

NAME: Archetypical-"Defined-Using"-slot
SPECIALIZATIONS:
SubSlots: Really-Defined-Using, Could-Have-Defined-Using
GENERALIZATIONS:
SuperSlots: Origin, **GoodConjecUnits**
Justification: H9
IS-A: Kind of slot
WORTH: 300
ORIGIN: Specialization of Origin
Defined-using: Specialize
Creation-date: 9/18/79 15:43
AVERAGE-SIZE: 1
FORMAT: Set
FILLED-WITH: Concepts
CACHE? Always-Cache
MAKES-SENSE-FOR: Concepts

Figure 21. Part of the concept containing centralizing knowledge about all DefinedUsing slots.

---

7. Heuristics used to develop new representations

The example here shows how new kinds of slots can be discovered and used to advantage. This is just an extension of a *given* representation, rather than true exploration in "the space of all representations of knowledge". I believe the latter will someday be possible, using nothing more than a body of heuristics for guidance, but we do not yet have enough experience to formulate the necessary rules.

Each kind of representation makes some set of operations efficient, often at the expense of other operations. Thus, an exploded-view diagram of a bicycle makes it easy to see which parts touch each other, sequential, verbal instructions make it easy to assemble the bicycle, an axiomatic formulation makes it easy to prove properties about it, etc.

As a field matures, its goals vary, its *paradigm* shifts, the questions to investigate change, the heuristics and algorithms to bring to bear on those questions evolve. Therefore, the utility of a given representation is bound to vary both from domain to domain and within a domain from time to time, much as did that of a given corpus of heuristics. The representation of today must adapt or give way to a new one -- or the field itself is likely to stagnate and be supplanted.

Where do these new representations come from? The most painless route is to merely select a new one from the stock of existing representational schemes. Choosing an appropriate representation means picking one which lets you quickly carry out the operations you're now going to carry out

most frequently.

In case there is no adequate existing representation, you may try to extend one, or devise a whole new one (good luck!), or (most frequently) simply employ *a set* of known ones, whose union makes all the common operations fast. Thus, when I buy a bicycle, I expect both diagrams and printed instructions to be provided. The carrying along of multiple representations simultaneously, and the concommitant need to shift from one to another, has not been much studied -- or attempted -- in AI to date, except in very tiny worlds (e.g., the Missionaries & Cannibals puzzle).

There are several levels at which "new representations" can be found. At the lowest level, one may say that AM changed its representation every time it defined a new domain concept or predicate, thereby changing its vocabulary out of which new ones could be built.

Much more significant would be the definition of new kinds of slots, typically ones specific to -- and very useful for -- some newly-discovered field of knowledge. For instance, when AM found the unique factorization conjecture, it would have been good if it had defined a new kind of slot, Prime-Factors, that every number could have had. A rule capable of this second-level representation augmentation is the following one:

IF the average size of *s* slots is large,
THEN propose a new task: replace *s* by new specializations of *s*.

The vague terms in the rule would have specific computational interpretations, of course; for instance, "large" might mean ">10", or "> 3 times the average size of all slots", or "larger than any other slot", or (most useful from a computational efficiency viewpoint) "larger than the average number of slots a unit has". It might cause the Examples slot to be broken into several subslots, such as ExtremeExamples, TypicalExamples, BoundaryExamples, etc. It might cause Factors to be split up into PrimeFactors, LargeFactors, etc. Note that the subslots will not in general be disjoint.

The third and final level at which "new representations" can be interpreted is to actually shift from one entire scheme to another -- perhaps novel -- one. The following two rules indicate when a certain type of shift is appropriate:

IF the problem is a geometric one,
THEN draw a diagram.

IF most units have most of their possible slots filled in,
THEN shift from property lists to record structures.

All the heuristics of this type are specializations of the general one which says IF some operation is performed frequently, THEN shift to a representation in which it is very inexpensive to perform.

Let us continue our example. Here is a heuristic which is capable of reacting to a situation by defining an entirely new slot, built up from old ones, a new slot which it expects will be useful:

H10: IF a slot *s* is very important, and all its values are units,
THEN-CREATE-NEW-KIND-OF-SLOT which contains "all the relations among the values of my *s* slot"

When the number stored in the Worth slot of the GoodConjecUnits concept is large enough, the system attends to the task of explicitly studying GoodConjecUnits. Several heuristics are relevant and fire; among them is H10, the rule shown above. It then synthesizes a whole new unit, calling it RelationsAmongEntriesOnMy"GoodConjecUnits"Slot. Every known way in which entries on the GoodConjecUnits slot of a concept C relate to each other will be recorded on this new slot of C.

For instance, take a look at the Primes concept (Figure 20). Its GoodConjecUnits slot contains the following entries: Times, Divisors-of, Exponentiation, Squaring, and Numbers-with-three-divisors. The first two of these entries are inverses of each others; that is, if you look over the Times unit,

you will see a slot called Inverse which is filled with names of concepts, including Times. Similarly, still looking over the Times unit, one can see a slot called Repeat which is filled with the entry Exponentiation, and one can see a slot called Compose filled with Squaring. So Inverse and Repeat and Compose are some of the relations connecting entries on the GoodConjecUnits slot of Primes, hence the program will record Inverse and Repeat and Compose as three entries on the RelationsAmongEntriesOnMy"GoodConjecUnits"Slot slot of the Primes concept.

Now it so happens that several concepts wind up with "Compose" and "Inverse" as entries on their RelationsAmongEntriesOnMy"GoodConjecUnits"Slot slot. The alert reader may suspect that this is no accident, and an alert program should suspect that, too. Indeed, the following heuristic says that it might be useful to behave as if "Compose" and "Inverse" were always going to eventually appear there:

H11: IF (for many units *u*) the *s* slot of *u* contains the same values v_i,
THEN-ADD-VALUE v_i to the *ExpectedEntries* slot of the *Typical-s-slot* unit.

This causes the program to add Compose and Inverse to the slot called ExpectedEntries of the concept called RelationsAmongEntriesOnMy"GoodConjecUnits"Slot. This one small act, the creation of a pair of links, is in effect creating a new heuristic which says:

IF a concept gets entries X and Y on its GoodConjecUnits slot,
THEN predict that it will get Inverse(X), Inverse(Y), and Compose(X,Y) there as well.

How is this actually used? Consider what occurs when the program defines a new concept, C, which is DefinedUsing Divisors-of. As soon as that concept is formed, the heuristic link from DefinedUsing to GoodConjecUnits automatically fills in Divisors-of as an entry on the GoodConjecUnits slot of C. Next, the links just illustrated above come into action, and place Inverse and Compose on the RelationsAmongEntriesOnMy"GoodConjecUnits"Slot slot of C. That in turn causes the inverse of Divisors-of, namely Times, to be placed on the GoodConjecUnits slot as well as the already-present entry, Divisors-of. Finally, that causes the program to go off looking for conjectures between C and either multiplication or division. When a conjecture comes in connecting C to one of them, it will get a higher a priori estimated worth than one which doesn't connect to them.

If only we'd had the new heuristics back when Primes was first defined, they would have therefore embodied enough "common sense" to prefer the Unique Factorization Theorem to Goldbach's conjecture. If we'd had them earlier, these heuristics would have led us to our present state much sooner. Because of our assumptions about the continuity of the world, such heuristics should nevertheless be useful from time to time in the future.

Notice that there's nothing special about mathematics -- the newly synthesized heuristics have to do with very general slots, like DefinedUsing and GoodConjecUnits. For instance, as soon as a new concept (say Middle-Class) is DefinedUsing Income, the program immediately fills in the following underlined information:

NAME: Middle-Class
Defined-using: Income
RelationsAmongEntriesOnMy"GoodConjecUnits"Slot: Inverse, Compose
Good-Conjec-Units: Income, Spending, EarnedInterest

Thus, it goes off looking for (and will expect more from) conjectures between Middle-Class and any of Income, Spending, and EarnedInterest. Thus the new slot *is* useful, though it has a terrible name, and the new little heuristics (which looked like little links or facts but were actually *permission to make daring guesses*) were powerful after all.

We have relied heavily on our representation being very structured; in a very uniform one (say a calculus of linear propositions, with the only operations being Assert and Match) it would be

difficult to obtain enough empirical data to easily modify that representation. This is akin to the nature of discovering domain facts and heuristics: if the domain is too simple, it's *harder* to find new knowledge and -- in particular -- new heuristics. Heuristics for propositional calculus are much fewer and weaker than those available for guiding work in predicate calculus; they in turn pale before the rich variety available for guiding theorem proving "the way mathematicians really do it". This is an argument for attacking seemingly-difficult problems which turn out to be lush with structure, rather than working in worlds so constrained that their simplicity has sterilized them of *heuristic structure.*

8. Conclusions

We began by noting that the limiting step in the construction of expert systems was building the knowledge base, and that one solution would be for the program itself to automatically acquire new knowledge: to learn via discovery.

The heuristic search paradigm seems adequate to guide a program in formulating useful new concepts, gathering data about them, and noticing relationships connecting them. However, as the body of domain-specific facts grows, the old set of heuristics becomes less and less relevant, less and less capable of guiding the discovery process effectively. New heuristics must also be discovered.

Since heuristics is a domain of knowledge, much like any other, one can imagine an expert system that works in that field. That is, a corpus of heuristics can grow and improve and gather data about itself. This process is very slow and explosive, yet it can be greatly facilitated by having "the right representation". In the case of a schematized representation, this means having the right set of slots or attributes, the right set of attached procedures, etc. We saw how heuristics can lead to the development of useful new kinds of slots, to improved representations of knowledge. It was hypothesized that the same representation we use for attributes and values of object-level concepts could also be used to represent heuristics and even to represent representation. To draw some examples from the RLL system [Lenat & Greiner 80]: Primes (a set of numbers), GeneralizeRarePredicate (a heuristic), GeneralizeRareHeuristic (a meta-heuristic), and Isa (a representation concept) are all represented adequately as units with slots having values. A single interpreter runs both meta-heuristics and heuristics, and is itself represented as a collection of units. While meta-heuristics *could* be tagged to distinguish them from heuristics, the *utility* of doing so rests on the existence of rules which genuinely treat them differently somehow -- and such rules have not to date been encountered.

One of the necessary steps in this research was the explication of at least a rudimentary theory of heuristics, an analysis of their innate source of power, their nature. This turned out to rest upon the continuity of our world: if the situation is very similar, so is the set of (in)appropriate actions to take. Corollaries of this provide the justification for the use of analogy and even for the utility of memory. The central assumption was seen to be just that -- an assumption which is often false in small ways, but which is nevertheless a useful fiction to be guided by.

By graphing the power curves of a heuristic (the utility of that heuristic as a function of task being worked on), we were able to see the gains -- and dangers -- of specializing and generalizing them to get new ones. Such curves determine a preferred order for obeying relevant heuristics, and suggest several specific new attributes worth measuring and recording for each heuristic (e.g., the sharpness with which it flips from useful to harmful, as one leaves its domain of relevance).

By arranging all the world's heuristics (well, at least all of AM's, and several more randomly-chosen ones from chess, biology, and oil spills) into a hierarchy using the relation "More-General-Than", we were surprised to find that hierarchy very shallow, thereby implying that analogy would be more useful a method of generating new heuristics than would specialization or generalization. By noting that both Utility and Task have several dimensions, most of this problem went away. By noting that two heuristics can have *many* important relations connecting them, of which More-General-

Than is just one example, the shallowness *problem* turns into a powerful heuristic: if a new heuristic *h* is to differ from an old one along some dimension (relation) *r*, then use analogy to get *h* if *r*'s graph is shallow, and use generalization/specialization if *r*'s graph is deep. We also discussed some useful slots which heuristics can have, and a method for generating new kinds of slots.

Before the research programme outlined in figure 2 can be completed, much more must be known about analogy, and more complete theories of heuristics and of representation must exist. Toward that goal we must obtain more empirical results from programs trying to find useful new domain-specific heuristics and representations.

References

Brown, John Seely, and Kurt VanLehn, "Repair Theory: A Generative Theory of Bugs in Procedural Skills," to appear in *J. Cog. Sci.*, IV, 4, 1980.

Clancey, William J., "Dialogue Management for Rule-Based Tutorials," *Proc. Sixth International Joint Conference on Artificial Intelligence*, Tokyo, 1979.

Feigenbaum, Edward A., "The Art of Artificial Intelligence," *Proc. Fifth International Joint Conference on Artificial Intelligence*, MIT, Boston, 1977.

Gaschnig, John, "Exactly How Good Are Heuristics?: Toward a Realistic Predictive Theory of Best-First Search", *Proc. Fifth International Joint Conference on Artificial Intelligence*, Cambridge, 1977.

Lenat, Douglas B., "On Automated Scientific Theory Formation: A Case Study Using the AM Program," in (Jean Hayes, Donald Michie, and L. I. Mikulich, eds.) *Machine Intelligence 9*, New York: Halstead Press, a division of John Wiley & Sons, 1979, pp. 251-283.

Lenat, Douglas B., and Russel D. Greiner, "RLL: A Representation Language Language," *Proc. of the First Annual Meeting of the American Association for Artificial Intelligence* (AAAI), Stanford, August, 1980.

Minsky, Marvin, "Steps Toward Artificial Intelligence", in (Feigenbaum and Feldman, eds.) *Computers and Thought*, McGraw-Hill, 1963.

Newell, Allen, and Herbert Simon, "Computer Science as Empirical Inquiry: Symbols and Search", *CACM*, 19, 3, March, 1976.

Poincare', H., *The Foundations of Science*. The Science Press, New York, reprinted in 1929.

Polya, G., *How to Solve It*, Princeton University Press, 1945.

Acknowledgements

Productive discussions with John Seely Brown, Bruce Buchanan, Bill Clancey, Johan deKleer, John Doyle, Russ Greiner, Mark Stefik, and Mike Williams have heavily influenced this work. Sections 3 and 4 are summaries of the lessons learned from AM, for which I thank Bruce Buchanan, Ed Feigenbaum, Cordell Green, Don Knuth, and Allen Newell. The data for Section 5's "shallowness" conclusion about the tree of heuristics was gathered while I was at CMU, with the aid of Herb Simon and Woody Bledsoe. Much of Sections 6 and 7 rely upon RLL, a self-describing and self-modifying representation language constructed by Russ Greiner and the author. Finally, I wish to thank XEROX PARC and Stanford's HPP for providing superb environments (intellectual, physical, and computational) in which to work. Financial support was provided by ONR (N00014-80-C-0609), NSF (MCS 79-0194), and XEROX.